Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Aleš Omerzel

Posplošena električna dominacija

diplomsko delo
univerzitetni študijski program prve stopnje
računalništvo in matematika

Mentor: prof. dr. Sandi Klavžar

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Diplomsko delo naj obravnava problem posplošene električne dominacije. Poseben poudarek naj bo na električni dominaciji na drevesih, implementiran naj bo tudi algoritem, ki v polinomskem času poišče optimalno električno dominantno množico.

Osnovna literatura:
G.J. Chang, P. Dorbec, M. Montassier, A. Raspaud, Generalized Power Domination of Graphs, Discrete Appl. Math. 160 (2012) 1691–1698.

The Faculty of Computer and Information Science issues the following thesis:

The thesis deals with the problem of the generalized power domination. Particular emphasis is put on the power domination on trees. An algorithm for finding the optimal power dominating set is implemented in polynomial time.

Primary bibliography:
G.J. Chang, P. Dorbec, M. Montassier, A. Raspaud, Generalized Power Domination of Graphs, Discrete Appl. Math. 160 (2012) 1691–1698.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleš Omerzel, z vpisno številko **63110362**, sem avtor diplomskega dela z naslovom:

Posplošena električna dominacija

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Sandija Klavžarja;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 3. september 2014 Podpis avtorja:

Iskreno se zahvaljujem mentorju, prof. dr. Sandiju Klavžarju, za vso pomoč pri izdelavi naloge in čas, ki mi ga je posvetil. Zahvaljujem se tudi mami Martini in očetu Ivanu, ki sta mi omogočila študij in mi z vso ljubeznijo in potrpljenjem stala ob strani. Hvala tudi lektorjem za pregled naloge in hvala prijateljem za vso moralno in ostalo pomoč.

Kazalo

Povzetek

Problem električne dominacije je optimizacijski problem, ki se je pojavil v novodobnem času skupaj z razvojem električnih omrežij. V električnem omrežju je potrebno nadzorovati tok in napetost na vseh stičiščih in povezavah. Merilne naprave so drage, zato vlada težnja h postaviti najmanjšega števila naprav tako, da bi omrežje ostalo v celoti nadzorovano. Problem električne dominacije se posploši na problem k-električne dominacije. V diplomski nalogi so z definicijo obeh problemov predstavljena pravila za širjenje nadzora v omrežju. Za tem je podan in dokazan linearen algoritem za iskanje optimalne k-električne dominantne množice na drevesih. Sledi implementacija algoritma v programskem jeziku Java. V zaključku pa je dokazana NP-polnost problema električne dominacije na dvodelnih grafih.

Ključne besede: električna dominacija, posplošena električna dominacija, drevo, Java, NP-poln problem.

Abstract

The power domination problem is an optimization problem that has emerged together with the development of the power networks. It is important to control the voltage and current in all the nodes and links in a power network. Measuring devices are expensive, which is why there is a tendency to place a minimum number of devices in a power network so that the network remains fully supervised. The k-power domination is a generalization of the power domination. The thesis represents the rules of the dissemination of the control in the network. A linear algorithm is provided for finding the optimal k-power dominating set on trees and followed by the implementation of the algorithm in Java. At the end NP-completeness for the power domination problem on bipartite graphs is proved.

Keywords: power domination, general power domination, tree, Java, NP-complete problem.

Poglavje 1

Uvod

Začetki elektrike segajo vse do starih Grkov, ki so poznali statično elektriko. Kos palice, ki je bil drgnjen ob krzno, je namreč privlačil krzno. Okoli leta 1800 je nato Alessandro Volta odkril baterijo, prvo javno oskrbo z električno energijo pa so leta 1881 s prižigom uličnih svetilk predstavili v mestu Surrey v Veliki Britaniji [6, 7, 8].

V 20. stoletju se je razvoj nadaljeval, začele so nastajati številne elektrarne, s tem pa so se pojavila velika električna omrežja. Z večanjem omrežij je bilo potrebno nadzorovati tok in napetost na kablih. Za merjenje toka in napetosti so obstajale fazne merilne naprave (PMU – phase measurement units), ki so predstavljale precejšen strošek za vzdrževalca omrežja. Da bi se stroški nabave in vgradnje v omrežje minimizirali, bi bilo idealno postaviti minimalno število teh naprav v omrežje. Ker obstajajo dodatna pravila, ki napravam omogočajo nadzirati tudi oddaljene točke in povezave, postane iskanje mest za postavitev najmanjšega števila naprav precej zahtevno. Od tod se je razvil problem električne dominacije.

Problem električne dominacije je splošnejša oblika klasičnega problema dominantne množice, kjer je potrebno na grafu G poiskati najmanjšo množico vozlišč $S \subseteq G$ tako, da so vsa vozlišča grafa G nadzorovana. Pri tem vozlišče v množici S nadzoruje sebe in vse svoje sosede [4].

Pri problemu električne dominacije iščemo najmanjšo množico vozlišč na enak način, vendar je to le prvi korak. Nadzor iz posameznega vozlišča se lahko tokrat tudi širi. Če so vsa sosednja vozlišča nadzorovanega vozlišča razen enega nadzorovana, postane nadzorovano tudi preostalo vozlišče.

k-električna dominacija se od preproste električne dominacije razlikuje v doda-

tnem parametru k, ki določa stopnjo posplošenosti problema. Natančneje, parameter k pove, največ koliko nenadzorovanih sosedov lahko ima neko nadzorovano vozlišče, da lahko širi nadzor med preostale nenadzorovane sosede.

V prvem poglavju bomo natančno definirali oba problema: električno dominacijo in k-električno dominacijo. Nato se bomo osredotočili na k-električno dominacijo in pokazali, v kakšnih povezavah so med sabo električne dominacije pri različnih parametrih k. Nato bomo na podlagi parametra k in velikosti grafa G podali omejitev za največjo možno optimalno k-električno dominantno množico.

Drugo poglavje namenjamo iskanju k-električne dominantne množice na drevesih. Predstavili bomo linearen algoritem in nato implementacijo v programskem jeziku Java. Pri implementacij bomo za grafični vmesnik uporabili knjižnico GraphStream.

V zadnjem poglavju pa dokažemo, da je problem električne dominacije NP-poln problem. Dokler je $NP \neq P$ pomeni, da je nesmiselno iskati algoritem, ki bi problem rešil v polinomskem času. Zato je bolje razviti kakšen aproksimacijski algoritem, hevristični algoritem, ali pa se poslužiti drugačnih metod.

Poglavje 2

Električna dominacija

V tem poglavju si bomo najprej ogledali osnovno električno dominacijo, ki ji bo sledila posplošena električna dominacija in več trditev v povezavi s splošno električno dominacijo. Najprej moramo definirati dve različni soseščini, ki ju bomo uporabljali v nadaljevanju.

Naj bo graf $G = (V, E)$, kjer je množica V množica vozlišč grafa G, množica E pa množica povezav grafa G. Množico vseh sosedov vozlišča v označimo z $N(v)$, torej $N(v) = \{u \in V(G) : uv \in E(G)\}$. Množici $N(v)$ pravimo *odprta soseščina* vozlišča v. Če odprti soseščini dodamo vozlišče v, dobimo *zaprto soseščino* $N[v]$, kjer je $N[v] = N(v) \cup \{v\}$.

2.1 Osnovna električna dominacija

Problem električne dominacije smo opisali že v uvodu. V tem poglavju bomo uvedli nove oznake in definirali nove pojme.

Označimo za *nadzorovano množico* množico vseh nadzorovanih vozlišč v določenem trenutku. Tedaj je širjenje nadzora na grafu določeno takole.

Definicija 2.1. [1, 2] *(Nadzorovana množica) Naj bo $G = (V, E)$ graf in $S \subseteq V(G)$. Nadzorovana množica $(P_G^i(S))_{i\geq 0}$ na koraku i je določena z začetnim pravilom P1 in širitvenim pravilom P2:*

- *P1: Za vsako vozlišče $v \in S$ so vozlišče v in vsa njegova sosednja vozlišča dodani v nadzorovano množico $P_G^0(S)$.*

- *P2: Če je vozlišče $v \in P_G^i(S)$ in je vozlišče u edino sosednje vozlišče vozlišča v, ki ni v nadzorovani množici $P_G^i(S)$, potem je vozlišče u dodano v nadzorovano množico v naslednji iteraciji: $u \in P_G^{i+1}(S)$.*

Opazimo, da se nadzorovana množica z vsako iteracijo poveča. Širjenje se zaključi, ko v drugem pravilu ne najdemo nadzorovanega vozlišča z enim nenadzorovanim sosedom, ki bi omogočal širjenje.

Definicija 2.2. [2] *(Električna dominantna množica) Množica S je* električna dominantna *množica takrat, kadar velja*

$$P_G^{\infty}(S) = V(G).$$

Najmanjši kardinalnosti takšne množice pravimo električno dominantntno število $\gamma_{\mathrm{P}}(G)$. $\gamma_{\mathrm{P}}(G)$-množica *je najmanjša električna dominantna množica grafa G.*

Če je $P_G^i(S) = V(G)$ za nek $i \geq 0$, potem s podano električno dominantno množico S uspešno nadzorujemo celoten graf G, vendar pri tem nimamo zagotovila, da je električna dominantna množica S tudi optimalna. Tako dobimo problem električne dominacije, kjer je potrebno poiskati najmanjšo električno dominantno množico S, da bo $P_G^{\infty}(S) = V(G)$.

2.2 Posplošena električna dominacija

Posplošena električna dominacija se od osnovne razlikuje v tem, da lahko vozlišče širi nadzor med svoje sosede, tudi če ima nenadzorovanih sosedov več kot enega.

Definicija 2.3. [2] *(Nadzorovana množica) Naj bo $G = (V, E)$ graf in $S \subseteq V(G)$. Nadzorovana množica $(P_G^i(S))_{i \geq 0}$ na koraku i je določena z začetnim pravilom P1 in širitvenim pravilom P2:*

- *P1: $P_G^0(S) = N[S]$,*
- *P2: $P_G^{i+1}(S) = \cup\{N[v] : v \in P_G^i(S)$ in $|N[v] \backslash P_G^i(S)| \leq k\}$.*

Definicija 2.4. [2] *(k-električna dominantna množica) Množica S je* k-električna dominantna množica*, če velja $P^{\infty}(S) = V(G)$. Najmanjšo kardinalnost takšne množice označimo z $\gamma_{\mathrm{P},k}(G)$. $\gamma_{\mathrm{P},k}(G)$-*množica *pa je množica z najmanjšo kardinalnostjo.*

Opazka 2.5. [2] *Vzemimo graf G in množici $S, S' \subseteq V(G)$. Če je $P^i(S) \subseteq P^j(S')$ za neki naravni števili i in j, je potem tudi $P^{i+1}(S) \subseteq P^{j+1}(S')$ in zato $P^\infty(S) \subseteq P^\infty(S')$. Torej, če je množica S tudi k-električna dominantna množica, je množica S' tudi k-električna dominantna množica.*

2.3 Povezava med k- in $(k+1)$-električno dominacijo

Poglejmo si, v kakšni povezavi so k-električne dominacije pri različnih vrednosti parametra k.

Trditev 2.6. [2] *Naj bo množica S tudi k-električna dominantna množica. Potem je množica S tudi k'-dominantna množica za vsak $k' \geq k$ in velja naslednja veriga neenakosti:*

$$\gamma(G) = \gamma_{\mathrm{P},0}(G) \geq \gamma_{\mathrm{P},1}(G) \geq \gamma_{\mathrm{P},2}(G) \geq \cdots$$

Dokaz. Najprej dokažimo trditev, da je množica S tudi k'-dominantna množica za vsak $k' \geq k$.

Če najprej vzamemo $i = 0$, dobimo

$$P^0_{G,k'}(S) = N[S] = P^0_{G,k}(S).$$

Za vsak naslednji korak i uporabimo pravilo P2. Če je pravilo P2 izpolnjeno za parameter k, sledi, da je izpolnjeno tudi za parameter k'. To vidimo iz pogoja

$$|N[v] \backslash P^i_G(S)| \leq k \leq k'.$$

Tedaj velja $P^i_{G,k}(S) \subseteq P^i_{G,k'}(S)$ za vsak i. Če množica S nadzoruje celoten graf, velja

$$V(G) = P^\infty_{G,k}(S) \subseteq P^i_{G,k'}(S).$$

Torej je $P^i_{G,k'}(S) = V(G)$ in zato je množica S tudi k'-električna dominantna množica.

□

Da neenakosti iz trditve 2.6 ne moremo izboljšati, sledi iz naslednjega rezultata.

Trditev 2.7. [2] *Naj bo* $\{x\}_{i=0}^{n}$ *končno nenaraščajoče zaporedje naravnih števil. Potem obstaja graf* G *tako, da je* $\gamma_{\mathrm{P},k}(G) = x_k$ *za vsak* k, $0 \leq k \leq n$.

Dokaz. Za vsak k, $0 \leq k \leq n$, vzamemo $x_k - x_{k+1}$ kopij grafa $K_{1,k+1}$ in x_{n+1} nastavimo na 0. Iz centrov zvezd sestavimo kliko tako, da vsak center povežemo z vsakim drugim centrom. Tedaj mora k-električna dominantna množica S vsebovati vsaj eno vozlišče iz vsake zvezde $K_{1,j}$, kjer je $j \geq k+1$. Recimo, da to ni res. Naj obstaja zvezda $K_{1,i}$, ki ne vsebuje vozlišča v množici S. Potem bo prvo nadzorovano vozlišče v zvezdi $K_{1,i}$ centralno vozlišče zvezde. Ker ima centralno vozlišče vsaj $k+1$ nenadzorovanih sosedov (listov zvezde $K_{1,i}$), ne more širiti nadzora med njimi. To je v protislovju s predpostavko, da je množica S tudi k-električna dominantna množica. Torej množica S res vsebuje vsaj eno vozlišče iz vsake zvezde $K_{1,j}$.

Naj bo število z enako številu zvezd $K_{1,j}$, za katere je $j \geq k+1$. Tedaj vemo, da je $\gamma_{\mathrm{P},k}(G) \geq z$. Pokažimo, da velja $\gamma_{\mathrm{P},k}(G) = z$. V množico S dodamo centre zvezd $K_{1,j}$. Označimo eno od teh vozlišč z v. Ker je vozlišče v v množici S, so po pravilu P1 nadzorovana vsa vozlišča v kliki, ki jo sestavljajo centri zvezd. V naslednji iteraciji širijo nadzor centri zvezd $K_{1,j'}$, kjer je $j' \leq k$. Torej smo pokrili celoten graf G in dobimo $\gamma_{\mathrm{P},k}(G) = z$. Za konec še natančno preštejmo število zvezd $K_{1,j}$:

$$z = (x_k - x_{k+1}) + (x_{k+1} - x_{k+2}) + \cdots + (x_n - x_{n+1}) = x_k - x_{n+1} = x_k.$$

Torej je $\gamma_{\mathrm{P},k}(G) = x_k$.

□

2.4 Implementacija grafa – zvezde v kliki

V tem razdelku je podana implementacija primera iz trditve 2.7.

Vhodni podatek za program je seznam nenaraščajočih pozitivnih celih števil *seq* in parameter k za stopnjo posplošenosti.

V rezultatu dobimo izrisan graf, ki ima označeno kliko in optimalno k-električno dominantno množico.

V dokazu trditve smo najprej definirali zvezde $K_{1,i}(G)$ in nato iz korenov zvezd sestavili kliko. Pri implementaciji smo ta vrstni red obrnili. Za začetek ustvarimo

nov razred *ConnectedStars*, ki mu nastavimo atribut *id*. Ta je potreben zgolj za kreiranje novih vozlišč in nima pomembnih funkcij. S funkcijo *createClique* zgradimo kliko. S funkcijo *compute* in *createStar* graf dopolnimo do ustreznih zvezd $K_{1,i}(G)$, ki bi nastale ob odstranitvi povezav klike. Vse komponente nato povežemo v glavnem programu *main*.

```
1  public static Node[] createClique ( Graph g, int size ) {
2
3      Node[] nodes = new Node[size];
4      for ( int i = 0; i < size; i++ ) {
5        Node n = g.addNode( Integer.toString( id += 1 ) ); // ustvari vozlisce
6        nodes[i] = n; // dodaj v seznam
7        for ( int j = 0; j < i; j++ ) { // dodaj povezave s preostalimi
       vozlisci
8          g.addEdge( Integer.toString( id += 1 ), nodes[j], n ); // dodaj
       povezovo
9        }
10     }
11     return nodes;
12   }
```

Program 2.1: Funkcija kreira kliko.

```
1  public static void compute ( Graph g, int[] seq, Node[] nodes, int domNum )
       {
2      int counter = 0;
3      // zanka po razlicnih vrstah zvezd
4      for ( int k = 0; k < seq.length - 1; k++ ) {
5        int dif = seq[k] - seq [k+1];
6        // kopije zvezd z enakim stevilom listov
7        for ( int j = 0; j < dif; j++ ) {
8          createStar( g, nodes[counter+j], 1, k+1 ); // funkcija, ki ustvari
       zvezdo
9          // pobarvaj center zvezde, ce le to ustreza pogoju
10         if ( k>= domNum ) {
11           nodes[counter+j].setAttribute( "ui.class", "required" );
12         }
13       }
14       // prestavi counter na zacetek naslednjega sklopa centrov
15       counter += dif;
16     }
17   }
```

Program 2.2: Funkcija za izdelavo pravilnega števila kopij zvezd.

```
1  public static void createStar ( Graph g, Node root, int length, int density
       ) {
2      for ( int i = 0; i < density; i++ ) {
3        Node parent = root;  // parent je vozlisce, ki bo dobilo novega soseda
4        for ( int j = 0; j < length; j++ ) {
5          Node child = g.addNode( Integer.toString( id += 1 ) ); // nov sosed
6          g.addEdge( Integer.toString( id += 1 ), parent, child ); // nova
       povezava
7          parent = child; // nastavi novo vozlisce
8        }
9      }
10   }
```

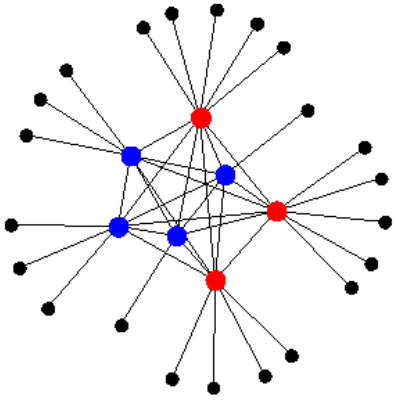
Program 2.3: Funkcija kreira zvezdo.

```
1  public static void main(String[] args) {
2
3      // nenarascujoce zaporedje pozitivnih celih stevil
4      int[] seq = {7, 5, 5, 3, 2, 0};
5      int k = 3;
6
7      // ustvari graf in ga inicializira
8      Graph g = new SingleGraph("Graph");
9      g.addAttribute("ui.stylesheet", styleSheet);
10     g.display();
11
12     Node[] nodes = createClique(g, seq[0]); // funkcija naredi kliko in vrne
        seznam vozlisc
13     colorizeClique (nodes); // funkcija pobarva vozlisca v kliki modro
14     compute(g, seq, nodes, k); // funkcija kreira zvezde, kjer so centri
       zvezd vozlisca iz klike
15   }
```

Program 2.4: Glavni program.

Primer, ki smo si ga izbrali, je enak primeru iz članka [2]. Za seznam smo vzeli števila $\{7, 5, 5, 3, 2, 0\}$ in $k = 3$. Ta dva parametra lahko v skladu s pravili poljubno spremenimo in tako dobimo drugačen rezultat.

Rezultat za primer v prejšnjem odstavku prikazuje slika 2.1. Rdeča vozlišča predstavljajo vozlišča v k-električni dominantni množici. Modra in rdeča vozlišča skupaj pa predstavljajo kliko.



Slika 2.1: Graf, kjer so centri zvezd med sabo povezani v kliko.

2.5 Meja za k-električno dominantno število

V nadaljevanju bomo spoznali nekaj trditev in lem, nato pa na podlagi števila vozlišč in parametra k v trditvi podali zgornjo mejo za k-električno dominantno število.

Trditev 2.8. [2] *Naj bo G povezan graf in naj bo največja stopnja grafa $\Delta(G) \leq k+1$. Potem velja*

$$\gamma_{\mathrm{P},k}(G) = 1.$$

Dokaz. Trdimo, da je množica $S = \{v\}$ tudi k-električna dominantna množica grafa G za vsako vozlišče v. Naj bo $w \in P_G^i(S)$ poljubno nadzorovano vozlišče na koraku i različno od v, $i \geq 0$. Vozlišče w mora imeti v $P_G^i(S)$ vsaj eno sosednje vozlišče, saj drugače vozlišče w ne bi moglo biti nazdorovano iz množice $S = \{v\}$. Zato je

$$|N(w) \backslash P_G^i(S)| \leq d(w) - 1 \leq k.$$

Druga neenakost velja po predpostavki $\Delta(G) \leq k+1$. Od tod sledi $N(w) \subseteq P^{i+1}(S)$. To pomeni, da lahko vsako vozlišče, ko postane nadzorovano, v naslednji iteraciji doda med nadzorovane še vse svoje sosede. Torej velja $P_G^{i+1}(S) = N[P_G^i(S)]$. Zaradi povezanosti grafa G pa dobimo

$$P_G^{\infty}(S) = V(G).$$

□

Grafi z maksimalno stopnjo $\Delta(G)$ manjšo ali enako $k+1$ teoretično niso posebej zanimivi, saj je dovolj v k-električno dominantno množico vstaviti eno poljubno vozlišče in graf bo v celoti nadzorovan. V nadaljevanju bomo zato obravnavali grafe z maksimalno stopnjo vsaj $k+2$. Takrat rešitev problema k-električne dominantne množice ni trivialno določena.

Trditev 2.9. [2] *Naj bo množica S tudi k-električna dominantna množica na grafu G. Če množica S vsebuje vozliše v stopnje največ $k+1$, potem je množica $S\backslash\{v\} \cup \{u\}$ tudi k-električna dominantna množica na grafu G za vsak $u \in N(v)$.*

Dokaz. Naj bo $S' = (S\backslash\{v\}) \cup \{u\}$. Ker je $u \in S'$ in $u \in N(v)$, po pravilu P1 k-električne dominacije velja, da je $u, v \in P^0(S')$. Ob uporabi predpostavke $d(v) \leq k+1$ je število nenadzorovanih sosedov vozlišča v v $P^0(S')$ največ k:

$$|N(v)\backslash P^0(S')| \leq d(v) - 1 \leq k.$$

Vozlišče v izpolnjuje pogoj širjenja iz pravila P2, zato je $N[v] \subseteq P^1(S')$. Torej velja $P^0(S) \subseteq P^1(S')$. Po opazki 2.5 je množica S' tudi k-električna dominantna množica na grafu G.

□

Trditev 2.10. [2] *Naj bo G povezan graf z največjo stopnjo vsaj $k+2$. Potem obstaja $\gamma_{\mathrm{P},k}(G)$-množica, ki vsebuje zgolj vozlišča stopnje vsaj $k+2$.*

Dokaz. Naj bo množica S $\gamma_{\mathrm{P},k}(G)$-množica, ki vsebuje največje možno število vozlišč stopnje vsaj $k+2$. Predpostavimo, da obstaja vozlišče $v \in S$, ki je stopnje največ $k+1$. Naj bo w vozlišče, ki je najbližje vozlišču v in je stopnje vsaj $k+2$. To pomeni, da je na poti med v in w minimalno število vozlišč in so vsa vozlišča stopnje največ $k+1$.

Vzemimo torej najkrajšo pot $p = (x_0, x_1, \ldots, x_j)$, kjer je $x_0 = v$ in $x_j = w$. Če uporabimo trditev 2.9 iterativno, je množica $S_i = S\backslash\{v\} \cup \{x_i\}$ k-električna dominantna množica na grafu G za vsak $i, 1 \leq i \leq j$. To je res tudi za $i = j$. Ker je $x_j = w$, dobimo v množici $S_j = S\backslash\{v\} \cup \{w\}$ eno vozlišče stopnje vsaj $k+2$ več kot v S. Vozlišče v s stopnjo največ $k+1$ namreč zamenjamo z vozliščem w,

ki pa je stopnje vsaj $k+2$. Torej smo v protislovju s predpostavko, da množica S vsebuje največje možno število vozlišč stopnje vsaj $k+2$.

□

Definicija 2.11. [2] S-privatna soseščina *vozlišča $v \in S$ je množica tistih sosednjih vozlišč vozlišča v, ki nimajo soseda v množici $S\backslash\{v\}$. S-privatno soseščino označimo z* $\mathrm{eps}(v, S)$ *(električna privatna soseščina).*

Trditev 2.12. [2] *Naj bo graf G povezan graf z maksimalno stopnjo vozlišča $\Delta(G) \geq k+2$. Potem obstaja $\gamma_{\mathrm{P},k}(G)$-množica S tako, da ima vsako vozlišče v v množici S vsaj $k+1$ S-privatnih sosedov: $|\mathrm{eps}(v, S)| \geq k+1$ za vsak $v \in S$.*

Dokaz. Naj bo množica S $\gamma_{\mathrm{P},k}(G)$-množica z vozlišči, ki imajo po trditvi 2.10 stopnjo vsaj $k+2$. Če za vsako vozlišče $v \in S$ velja $|\mathrm{eps}(v, S)| \geq k+1$, potem trditev velja.

Predpostavimo sedaj, da obstaja vozlišče $v \in S$ tako, da je $|\mathrm{eps}(v, S)| \leq k$. Obravnavajmo dve možnosti.

1) Naj bo vozlišče $w \in S$ sosednje vozlišču v. Trdimo, da je množica $S' = S\backslash\{v\}$ tudi k-električna dominantna množica na grafu G. Ker je $w \in S$, je $v \in P^0(S')$ in po predpostavki $|\mathrm{eps}(v, S)| \leq k$ dobimo

$$|N(v)\backslash P^0(S')| = |\mathrm{eps}(v, S)| \leq k.$$

Dalje velja $N[v] \subseteq P^1(S')$ in zato je $P^0(S) \subseteq P^1(S')$. Po opazki 2.5 je množica S' tudi k-električna dominantna množica na G. To pa je v protislovju z minimalnostjo množice S, saj je

$$|S'| = |S\backslash\{v\}| < |S|.$$

2) Naj bo vozlišče $v \in S$ brez soseda v množici S. Ker je $d(v) \geq k+2$ in je moč S-privatne soseščine vozlišča v enaka $|\mathrm{eps}(v, S)| \leq k$, obstaja vozlišče $w \in N(v)$ in vozlišče $x \in S$ tako, da je vozlišče $w \in N(x)$. Vozlišče w zaradi vozlišča v ni privatni sosed vozlišča x.

Naj bo

$$S' = (S\backslash\{v\}) \cup \{w\},$$

če je $d(w) \geq k+2$, sicer naj bo

$$S' = S\backslash\{v\}.$$

Sedaj trdimo, da je množica S' tudi k-električna dominantna množica. V obeh primerih velja

$$v \in N(w) \subseteq P^1(S').$$

V prvem primeru je vozlišče $w \in S'$ in zato je

$$v \in P^0(S') \subseteq P^1(S').$$

V drugem primeru imamo $d(w) \leq k+1$ in vozlišče $x \in S'$. Zato lahko uporabimo širitveno pravilo P2. Če sedaj upoštevamo predpostavko $|\mathrm{eps}(v,S)| \leq k$, velja

$$|N(v) \backslash P^1(S')| \subseteq |\mathrm{eps}(v,S)| \leq k.$$

Od tod sledi, da lahko vozlišče v v naslednji iteraciji razširi nadzor na svoje sosede. Torej je $N[v] \subseteq P^2(S')$ in zato dobimo $P^0(S) \subseteq P^2(S')$. Po opazki 2.5 sedaj sledi, da je množica S' tudi k-električna dominantna množica. Ker ima množica S' eno vozlišče manj kot množica S, ki je po definiciji optimalna množica, smo prišli do protislovja.

Torej ima vsako vozlišče v optimalni k-električni dominantni množici S vsaj $k+1$ S-privatnih sosedov.

□

Prišli smo do glavne trditve v tem poglavju. Poiskati želimo ostro zgornjo mejo za velikost optimalne k-električne dominantne množice. Poznavanje zgornje meje nam pomaga pri iskanju rešitve. Če je množica S tudi k-električna dominantna množica in je velikost množice S večja od zgornje meje, lahko sklepamo, da množica S zagotovo ni optimalna rešitev. Če pa je velikost množice S pod zgornjo mejo, smo optimalni rešitvi relativno blizu. Možno je, da smo optimalno rešitev tudi našli, vendar nam v splošnem zgornja meja ne da tega zagotovila.

Izrek 2.13. *Naj bo G povezan graf reda $n = |V| \geq k+2$. Potem velja*

$$\gamma_{\mathrm{P},k}(G) \leq \frac{n}{k+2}. \tag{2.1}$$

Meja je najboljša možna.

Dokaz. Veljavnost neenakosti 2.1 dobimo neposredno iz trditve 2.8 in trditve 2.12. Če je $\Delta(G) \leq k+1$, po trditvi 2.8 velja $\gamma_{\mathrm{P},k}(G) = 1$. Po predpostavki izreka 2.13 imamo $n \geq k+2$, zato dobimo

$$\gamma_{\mathrm{P},k}(G) = 1 \leq \frac{n}{k+2}.$$

Če je $\Delta(G) \geq k+2$, po trditvi 2.12 obstaja $\gamma_{\mathrm{P},k}(G)$-množica S, ki vsebuje samo vozlišča z vsaj $k+1$ S-privatnimi sosedi. Od tod sledi, da je število vseh privatnih sosedov množice S enako

$$\sum_{i=1}^{|S|}(k+1) = |S|(k+1).$$

Zraven prištejemo še vsa vozlišča iz množice S in vsota le-teh ne more prerasti števila vseh vozlišč v grafu G:

$$|S|(k+1) + |S| = |S|(k+2) \leq |V(G)| = n$$

oziroma

$$|S|(k+1) + |S| \leq |V(G)|.$$

Ker je $\gamma_{\mathrm{P},k}(G) \leq |S|$, dobimo

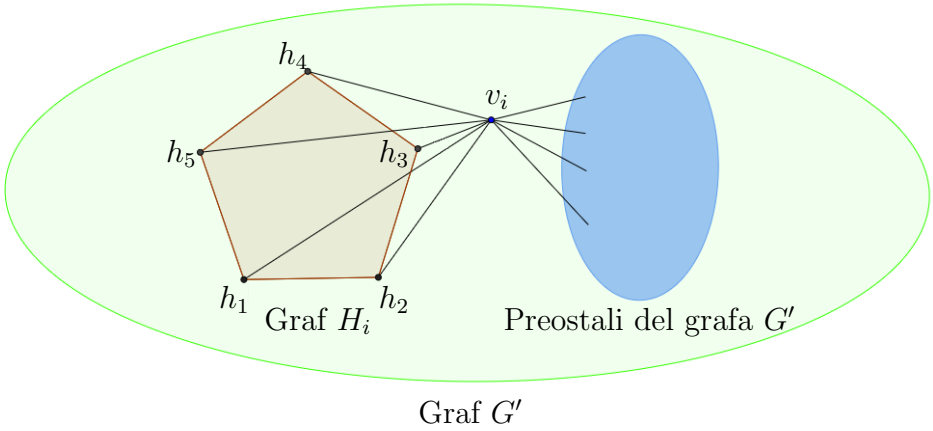
$$\gamma_{\mathrm{P},k}(G)(k+2) \leq |S|(k+2) \leq n$$

in od tod $\gamma_{\mathrm{P},k}(G) \leq \frac{n}{k+2}$.

Poiskati moramo še primer, s katerim dosežemo zgornjo mejo. Naj bo G povezan graf reda n. Vozlišča grafa G označimo z $v_1, v_2, \ldots, v_n$. Naj bo $H_1, H_2, \ldots, H_n$ družina grafov na $k+1$ vozliščih. Nov graf G' zgradimo tako, da vzamemo disjunktivno unijo teh grafov in dodamo povezave tako, da vozlišče v_i povežemo z vsemi vozlišči v H_i za vsak i, $1 \leq i \leq n$.

Pokažimo, da mora k-električna dominantna množica grafa G' vsebovati vsaj eno vozlišče iz množice vozlišč $V(H_i) \cup \{v_i\}$. Dokazujemo s protislovjem. Recimo, da obstaja število j, $1 \leq j \leq n$, in k-električna dominantna množica S, ki ne vsebuje vozlišča iz množice $V(H_j) \cup \{v_j\}$.

Na začetku, pred uporabo pravil P1 in P2, v množici H_j po predpostavki ni nadzorovanega vozlišča, zato je $P^0(S) \cap V(H_j) = \emptyset$. Naj bo $i \in \mathbb{N}$ takšno najmanjše število, da obstaja vozlišče $x \in V(H_j) \cap P^i(S)$. Vozlišče x in število



Slika 2.2: Primer grafa G' z vozliščem v_i ter podgrafom H_i na petih vozliščih.

i zagotovo obstajata, saj je množica S tudi k-električna dominantna množica, ki lahko s pomočjo pravil P1 in P2 nadzoruje celoten graf. To pomeni, da od i-te iteracije naprej nadzorujemo tudi x. Ker je vozlišče x dodano v nadzorovano množico $P^i(S)$, obstaja sosed $y \in N(x)$, za katerega velja $y \in P^{i-1}(S)$. Vozlišče x ima natanko $k+1$ sosedov, vozlišče v_j ter k vozlišč v H_j. Ker je eno od teh vozlišče y, ki je nadzorovano, ima vozlišče x največ k sosedov, ki niso v $P^{i-1}(S)$. Ker je $N(x) \subseteq V(H_j) \cup \{v_j\}$ in $V(H_j) \cap P^{i-1}(S) = \emptyset$, sledi $v_j = y \in P^{i-1}(S)$. Če pogledamo vozlišče v_j, velja $N[v_j] \backslash P^{i-1}(S) \supseteq V(H_j)$. Zato je

$$|N[v_j] \backslash P^{i-1}(S)| \geq |V(H_j)| = k+1.$$

To pomeni, da vozlišča v grafu H_j ne morejo biti nadzorovana, saj širitvenega pravila P2 na vozlišču v_j ne moremo uporabiti. Število nenadzorovanih sosedov vozlišča v_j je namreč več kot k. Tako pridemo do protislovja.

Torej mora k-električna dominantna množica grafa G' vsebovati vsaj eno vozlišče iz množice $V(H_i) \cup \{v_i\}$ za vsak $i, 1 \leq i \leq n$. To pomeni, da k-električna dominantna množica izmed vozlišč v $V(G')$ vsebuje vsaj n vozlišč. Ker je $|V(G')| = n(k+2)$ in $\gamma_{\mathrm{P},k}(S) = n$, v trditvi dosežemo enakost

$$\gamma_{\mathrm{P},k}(G') = n = \frac{n(k+2)}{k+2} = \frac{|V(G')|}{k+2}.$$

□

Poglavje 3

Električna dominacija na drevesih

Rešitev problema k-električne dominacije je v splošnem NP-poln problem. Mi si bomo ogledali iskanje k-električne dominantne množice na drevesih. Predstavili bomo algoritem ter implementacijo algoritma v programskem jeziku Java.

Problem k-električne dominacije na drevesih je linearne časovne zahtevnosti. Algoritme za iskanje k-električne dominantne množice najdemo v [2, 4, 5].

Predstavili bomo algoritem iz članka [2]. Glavna ideja je postopno brisanje listov. V ta namen za vsako vozlišče uvedemo posebne oznake, ki nosijo dodatne informacije. Algoritem jih uporablja za svoje izvajanje in določanje k-električne dominantne množice.

3.1 Predpriprave

Preden se lotimo algoritma, najprej vsakemu vozlišču dodamo dve oznaki: a_v, b_v. Oznaka a_v lahko zavzame tri vrednosti: $a_v \in \{R, F, B\}$. Oznaka R (required) označuje vozlišče v k-električni dominantni množici. Oznaka F (free) označuje nadzorovano vozlišče, ki lahko širi nadzor, in oznaka B (bound) označuje nenadzorovano vozlišče. V nadaljevanju bomo rekli, da je vozlišče v v stanju R, če je $a_v = R$, in enako za stanji F in B [2].

Druga oznaka je b_v. To je število, ki se uporablja predvsem v stanjih F in

B. Oznaka pove, h koliko sosedom lahko pripadajoče vozlišče širi nadzor. Pri k-električni dominaciji vrednost oznake ne preseže vrednosti k [2].

Poglejmo si definicijo L-električne dominantne množice, ki je splošnejša oblika definicije k-električne dominantne množice.

Definicija 3.1. *[2] Vsakemu vozlišču grafa G pripnemo oznako $L(v) := (a_v, b_v)$, kjer je $a_v \in \{R, F, B\}$ in $b_v \in \{0, 1, \ldots, k\}$. L-električna dominantna množica S grafa G je podmnožica množice vozlišč $V(G)$ tako, da za vsako vozlišče v z oznako $a_v = R$ velja $v \in S$. Nadzorovana množica $P^i_{G,L}(S)$ na koraku i, $i \geq 0$, je definirana po naslednjih dveh pravilih:*

(PL1) $P^0_{G,L}(S) = N[S] \cup \{v : a_v = F\}$,

(PL2) $P^{i+1}_{G,L}(S) = P^i_{G,L}(S) \cup \{N[v] : v \in P^i_{G,L}(S)$ *in* $|N[v] \backslash P^i_{G,L}(S)| \leq b_v\}$.

Začetno nadzorovano množico L-električne dominantne množice inicializiramo z $N[S]$, kar je enako kot pri k-električni dominaciji. Dodamo pa še vozlišča v stanju F. Posploši se tudi širitveni pogoj. V k-električni dominaciji je pogoj širjenja nadzora iz nadzorovanega vozlišča v izpolnjen, če je število nenadzorovanih sosedov $|N[v] \backslash P^i_{G,L}(S)|$ manjše od k. Število k je skupno vsem vozliščem. V L-električni dominaciji pa je pogoj širnjena nadzora določen z oznako b_v za vsako vozlišče posebej. Če vsakemu vozlišču $v \in V(G)$ nastavimo $L(v) = (B, k)$, dobimo ravno k-električno dominacijo.

Kadar bo vozlišče $v \in P^i_{G,L}(S)$ zadoščalo pogoju $|N[v] \backslash P^i_{G,L}(S)| \leq b_v$, bomo rekli, da vozlišče v zadošča širitvenemu pogoju L-PC (L-propagation condition). Z $\gamma_{\mathrm{P},L}(G)$ bomo označili L-električno dominantno število, ki je definirano kot najmanjša kardinalnost L-električne dominantne množice na grafu G. $\gamma_{\mathrm{P},L}(G)$-množica pa naj bo optimalna L-električna dominantna množica na grafu G.

Sedaj je za algoritem vse pripravljeno.

3.2 Algoritem

Algoritem Naj bo G drevo in L oznaka vozlišč: $L(v) := (a_v, b_v)$. Naj bo vozlišče x list drevesa G in vozlišče y sosednje vozlišče vozlišča x. Naj bo $G' = G - x$ in L' funkcija oznak na množici vozlišč $V' = V \backslash \{x\}$. Velja naslednje:

(1) Če je $a_x = R$, potem je $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G') + 1$ in $a'_y = F$, kjer je $a_y = B$. Sicer za trditve od (2) do (5) predpostavimo $a_x \neq R$.

(2) Če je ($a_y = R$) ali ($a_x = F$ in $b_x = 0$), potem je $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$.

(3) Če je $a_x = B$ in $b_y > 0$, potem je $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$ in $b'_y = b_y - 1$.

(4) Če je $a_x = B$ in $b_y = 0$, potem je $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$ in a'_y postane R.

(5) Če je $a_x = F$, $b_x > 0$ in $a_y \neq R$, potem je $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$ in $a'_y = F$.

Dokaz. (1) Naj bo S $\gamma_{\mathrm{P},L}(G)$-množica. Ker je $a_x = R$, je vozlišče x v dominantni množici S. Naj bo množica $S' = S\backslash\{x\}$. Z indukcijo pokažimo, da za korak i velja $P^i_{G',L'}(S') = P^i_{G,L}(S)\backslash\{x\}$. Opomnimo, da je vozlišče x list.

Baza indukcije: Na koraku $i = 0$ imamo v množici $P^0_{G,L}(S)\backslash\{x\}$ vsa vozlišča iz množice S ter njihove sosede. Odstraniti moramo le vozlišče x, medtem ko edino sosednje vozlišče y ostaja v nadzorovani množici. V nadzorovani množici $P^0_{G',L'}(S')$ je vse enako, razen pri vozlišču x in njegovemu edinemu sosedu y, saj vozlišče $x \notin S'$. Ker pa je $a_y \neq B$, je vozlišče y nadzorovano. Torej je $P^0_{G',L'}(S') = P^0_{G,L}(S)\backslash\{x\}$.

Indukcijska predpostavka: Naj bo sedaj enakost $P^i_{G',L'}(S') = P^i_{G,L}(S)\backslash\{x\}$ resnična do koraka $i \geq 0$.

Indukcijski korak: Naj bo vozlišče $v \in P^i_{G,L}(S)\backslash\{x\}$. Po indukcijski predpostavki je vozlišče $v \in P^i_{G',L'}(S')$. Dalje velja $N_G[v]\backslash P^i_{G,L}(S) = N_{G'}[v]\backslash P^i_{G',L'}(S')$. V enakosti bi edini problem povzročili vozlišči x in y, ki pa tako nista vsebovani niti na levi niti na desni strani enakosti. Zato enakost velja. Od tod dobimo $b'_v = b_v$ za vse $v \in P^i_{G',L'}(S')$. To velja tudi, če je $v = y$. Povezava xy je namreč nadzorovana, ali pa je ni, zato ne dodaja vrednosti oznaki b_y. Torej sledi, da vozlišče v zadošča L-PC na G natanko tedaj, ko zadošča L'-PC na G'.

Od tod sledi, da je $P^i_{G',L'}(S') = P^i_{G,L}(S)\backslash\{x\}$ za vsak i in zato je množica S' tudi L'-električna dominantna množica na grafu G'. Ker je $S' = S\backslash\{x\}$, kjer je množica S optimalna L-električna dominantna množica, velja $\gamma_{\mathrm{P},L}(G) \geq \gamma_{\mathrm{P},L'}(G') + 1$.

Ugotovili smo, da je kardinalnost množice S' vsaj za eno manjša od kardinalnosti optimalne L-električne dominantne množice S. Zanima nas, ali je to optimum.

Naj bo sedaj množica S' tudi $\gamma_{\mathrm{P},L'}(G')$-množica. Naj bo $S = S'+\{x\}$. Podobno velja $P^i_{G,L}(S) = P^i_{G',L'}(S')\cup\{x\}$ za vsak i. Vozlišče x ne vpliva na širjenje nadzora, ker se edinemu sosedu y oznaka b_y ne spremeni: $b_y = b'_y$. Vozlišče x poskrbi le

zase, da postane nadzorovano. Tako je množica S tudi L-električna dominantna množica na G. Torej velja, da je kardinalnost optimalne L-električne dominantne množice na grafu G največ za 1 večja od kardinalnosti optimalne množice S': $\gamma_{\mathrm{P},L}(G) \leq \gamma_{\mathrm{P},L'}(G') + 1$. Od tod sledi

$$\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G') + 1.$$

Obdelali smo primer, ko je $a_x = R$. V nadaljevanju bomo privzeli $a_x \neq R$. Naj bo S $\gamma_{\mathrm{P},L}(G)$-množica. Predpostavimo, da vozlišče $x \notin S$. V nasprotnem primeru lahko vzamemo množico $S' = S \backslash \{x\} \cup \{y\}$. Torej vozlišče x, ki je list na drevesu G, v množici S zamenjamo s staršem y. Ker velja $P_L^0(G) \subseteq P_L^0(S')$, je množica S' po opazki 2.5 tudi L-električna dominantna množica. Množici S in S' sta istega reda, zato je množica S' prav tako optimalna množica.

(2) Če je $a_y = R$, postane vozlišče x na grafu G nadzorovano v začetni iteraciji $i = 0$. Za širjenje nadzora na ostalih vozliščih je to enako, kot da bi vozlišče x odstranili z grafa G, saj je b_y v obeh primerih enak: $b_y = b'_y$. Z odstranitvijo vozlišča x dobimo ravno graf G'. Če še upoštevamo, da vozlišče x ni v L-električni dominantni množici, sledi $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$.

Preostane nam še drugi del, kjer je $a_x = F$ in $b_x = 0$.

Naj bo množica $S \subseteq V(G) \backslash \{x\}$ množica, ki ne vsebuje vozlišča x. Trdimo, da je

$$P^i_{G',L'}(S) \cup \{x\} = P^i_{G,L}(S)$$

za vsak i, $i \geq 0$. Dokažimo z indukcijo.

Baza indukcije: Če upoštevamo $a_x = F$, po uporabi pravila PL1 enakost za $i = 0$ velja: $P^0_{G',L'}(S) \cup \{x\} = P^0_{G,L}(S)$. Natančneje, razmislek je enak razmisleku v bazi indukcije primera (1).

Indukcijska predpostavka: Predpostavimo, da enakost $P^i_{G',L'}(S) \cup \{x\} = P^i_{G,L}(S)$ velja do koraka $i \geq 0$.

Indukcijski korak: Vsako vozlišče, ki zadošča L'-PC na grafu G' na koraku i, zadošča tudi L-PC na grafu G in obratno. Edino vozlišče, za katerega to mogoče ne velja, je vozlišče y. Vozlišče y lahko zadošča pogoju L'-PC na G', vendar ne zadošča pogoju L-PC na grafu G, saj ima na grafu G enega soseda več. Dodatni sosed je vozlišče x. Ker pa je $a_x = F$, je po pravilu PL1 vozlišče $x \in P^0_{G,L}(S) \subseteq P^i_{G,L}(S)$

in zato

$$N_G[y]\backslash P^i_{G,L}(S) \subseteq N_{G'}[y]\backslash P^i_{G',L'}(S).$$

Torej tudi vozlišče y zadošča pogoju L-PC tedaj, ko zadošča pogoju L'-PC.

Pokazali smo, da vsako vozlišče, razen vozlišče x, zadošča pogoju L'-PC natanko tedaj, ko zadošča pogoju L-PC. Ker je $b_x = 0$, je vozlišče x nepomembno tako za L'-PC kot tudi za L-PC. Sedaj upoštevamo še indukcijsko predpostavko ter dobimo

$$P^{i+1}_{G,L}(S) = P^{i+1}_{G',L'}(S) \cup \{x\}.$$

(3) Naj bo S $\gamma_{\mathrm{P},L}(G)$-množica, ki ne vsebuje vozlišča x. Trdimo, da je množica S tudi L'-električna dominantna množica na grafu G'. Najprej pokažimo, da za vsak $i \geq 0$ velja

$$P^i_{G,L}(S)\backslash\{x\} \subseteq P^i_{G',L'}(S).$$

Baza indukcije: Pri $i = 0$ uporabimo pravilo PL1 na grafu G in na grafu G'. Dobimo nadzorovani množici $P^0_{G,L}(S)$ ter $P^0_{G',L'}(S)$. Obe vsebujeta množico S in sosednja vozlišča množice S: $N(S)$. Če vozlišče $x \notin P^0_{G,L}(S)$, potem sta nadzorovani množici enaki. Če je vozlišče $x \in P^0_{G,L}(S)$, je potem $x \in N(S)$ zaradi $a_x \neq R$ in dobimo $P^0_{G,L}(S)\backslash\{x\} = P^0_{G',L'}(S)$. Torej trditev velja.

Indukcijska predpostavka: Predpostavimo, da velja inkluzija $P^i_{G,L}(S)\backslash\{x\} \subseteq P^i_{G',L'}(S)$ do koraka $i \geq 0$.

Indukcijski korak: Naj poljubno vozlišče $v \neq x$ zadošča pogoju L-PC na grafu G. To pomeni, da izpolnjuje pogoje širjenja nadzora k preostalim svojim sosedom. Če je $v \neq y$, velja $b_v = b'_v$ in zato vozlišče v zadošča tudi L'-PC na grafu G'.

Pokažimo, da velja enako za vozlišče y. Torej predpostavimo, da vozlišče y zadošča pogoju L-PC in glede na vozlišče x obravnavajmo dve možnosti.

A) Naj bo vozlišče $x \in P^i_{G,L}(S)$. Torej je vozlišče x že nadzorovano in je zato v nazdorovani množici. Ponovno ločimo dve množnosti glede na to, kako vozlišče x postane nadzorovano:

(I) Če velja $x \in P^0_{G,L}(S)$, pomeni, da je $y \in S$. Zato je $N[y] \subseteq P^0_{G,L}(S)$.

(II) Sicer je vozlišče x bilo dodano v nadzorno množico kasneje. To pomeni, da je vozlišče y zadoščalo pogojem širjenja, torej pogoju L-PCju na enem od prejšnjih korakov $j < i$.

V obeh primerih je $N[y] \subseteq P^i_{G,L}(S)$. Ker je $N_{G'}[y] \subseteq N_G[y]$, po indukcijski predpostavki velja $N_{G'}[y] \subseteq P^i_{G',L'}(S')$. Torej vozlišče y zadošča pogoju L'-PC na grafu G'.

Do sedaj smo ugotovili, da vozlišče y zadošča pogoju L'-PC, če zadošča pogoju L-PC in je $x \in P^0_{G,L}(S)$.

B) Poglejmo sedaj drugo možnost za vozlišče x: $x \notin P^i_{G,L}(S)$. Tedaj velja

$$N_G[y] \backslash P^i_{G,L}(S) = N_{G'}[y] \backslash P^i_{G',L'}(S) \cup \{x\}.$$

Zato iz pogoja $|N_G[y] \backslash P^i_{G,L}(S)| \leq b_y$ (iz pravila PL2) sledi, da je

$$|N_{G'}[y] \backslash P^i_{G',L'}(S)| \leq b_y - 1 = b'_y$$

v grafu G', saj vozlišče y v grafu G' ne vsebuje vozlišča x. Torej, če vozlišče y zadošča pogoju L-PC, zadošča tudi pogoju L'-PC. Tukaj upoštevamo tudi predpostavko $b_y > 0$, ker mora biti $b'_y \geq 0$.

Pripomnimo, če vozlišče x zadošča pogoju L-PC na grafu G, potem mora zaradi pogoja v PL2 veljati $x \in P^i_{G,L}(S)$. Vozlišče x pa je lahko postalo nadzorovano zgolj preko vozlišča y, zato je tudi $y \in P^i_{G,L}(S)$. Torej ta primer ni pomemben za posebno obravnavo, saj je že vključen v primer A. To zaključi naš dokaz trditve, da je množica S tudi L'-električna dominantna množica na grafu G'.

Naj bo sedaj S' $\gamma_{\mathrm{P},L'}(G')$-množica. Najprej opazimo, da za vsak korak $i \geq 0$ velja $P^i_{G',L'}(S') \subseteq P^i_{G,L}(S')$. Če na desni strani grafu G odvzamemo vozlišče x in ustrezno popravimo označevanje L, dobimo enakost. Pri tem imamo v mislih, da je vozlišče x še vedno list drevesa. Torej do edinega problema pridemo, če na nekem koraku i vozlišče y zadošča pogoju L'-PC na grafu G' in ne zadošča pogoju L-PC na grafu G. Do tega lahko pride, ker ima vozlišče y na grafu G enega soseda več kot na grafu G' : $N_G[y] \backslash N_{G'}[y] = \{x\}$. Da bi omenjeni problem nastal, bi moralo veljati $b_y \leq b'_y$. Ker pa med b_y in b'_y velja zveza $b_y = b'_y + 1$, do problema ne pridemo.

Odkar je množica S' tudi L'-električna dominantna množica na grafu G', mora na nekem koraku vozlišče y zadoščati pogoju L'-PC na grafu G'. Ker velja $P^i_{G',L'}(S') \subseteq P^i_{G,L}(S')$, vozlišče v zadošča tudi pogoju L-PC na grafu G. Tako vozlišče x postane nadzorovano na naslednjem koraku.

(4) Če je $a'_y = R$, potem poljubna L'-električna dominantna množica S' na grafu G' vsebuje vozlišče y. Zato je množica S' tudi L-električna dominantna množica na grafu G, saj vozlišče y nadzoruje tudi vozlišče x. Od tod sledi $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$, saj lahko za množico S' vzamemo tudi optimalno L'-električno dominantno množico na grafu G'.

Naj bo S $\gamma_{\mathrm{P},L}(G)$-množica, ki ne vsebuje vozlišča x. S protislovjem dokažimo, da mora veljati $y \in S$. Recimo, da $y \notin S$. Potem velja naslednje. Če vozlišče $x \notin P^0_{G,L}(S)$, vozlišče y zaradi $b_y = 0$ nikoli ne ustreza pogoju L-PC. V tem primeru vozlišče x ne bo nikoli nadzorovano in pridemo v protislovje s predpostavko, da je S $\gamma_{\mathrm{P},L}(G)$-množica. Torej je vozlišče $x \in P^0_{G,L}(S)$. Ker je $a_x = B$, mora biti vozlišče $x \in N(S)$. Ker je $N(x) = \{y\}$, mora biti vozlišče $y \in S$. Prišli smo v protislovje s predpostavko, da $y \notin S$. Torej je $y \in S$.

Od tod sedaj velja $a'_y = R$ in $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$.

(5) Naj velja $a_x = F$, $b_x > 0$ in $a_y \neq R$. Za vsako množico S, ki ne vsebuje vozlišča x, trdimo, da je

$$P^i_{G,L}(S) \subseteq P^i_{G',L'}(S) \cup \{x\} \subseteq P^{i+1}_{G,L}(S).$$

Od tod bi sledilo, da je množica S tudi L-električna dominantna množica na grafu G natanko tedaj, ko je množica S tudi L'-električna dominantna množica na grafu G'.

Za $i = 0$ prva inkluzija velja, saj je

$$P^0_{G,L}(S) = N[S] \cup \{v : a_v = F\} = N[S] \cup \{v : a_v = F, v \neq x\} \cup \{x\} \subseteq$$

$$\subseteq N[S] \cup \{v : a'_v = F, v \neq x\} \cup \{x\} = P^0_{G',L'}(S) \cup \{x\}.$$

Ker je $a'_y = F$ in je tudi pogoj L-PC enak L'-PC na preostalih vozliščih $v \notin \{x, y\}$, velja inkluzija tudi za vsak naslednji korak i:

$$P^i_{G,L}(S) \subseteq P^i_{G',L'}(S) \cup \{x\}.$$

Za drugo inkluzijo opazimo, da je $P^0_{G',L'}(S) \backslash \{y\} \subseteq P^0_{G,L}(S)$, saj je $a_y = B$ in $a'_y = F$. Enakost $a_y = B$ ugotovimo iz $b_x > 0$. Ker je $x \in P^0_{G,L}(S)$ in x zadošča pogojem L-PC na grafu G, je vozlišče $y \in P^1_{G,L}(S)$. Druga inkluzija torej velja za $i = 0$. Vozlišče y je torej edino vozlišče, ki je en korak kasneje v

nadzorovani množici $P^0_{G,L}(S)$ kot v nadzorovani množici $P^0_{G',L'}(S)$. Zato so tudi sosednja vozlišča vozlišča y največ za eno iteracijo kasneje v nadzorovani množici. Od tod sledi, da vsako vozlišče, ki zadošča L'-PC na grafu G' na koraku i, zadošča tudi L-PC na grafu G na koraku i ali pa $i+1$. Torej velja $P^i_{G',L'}(S) \cup \{x\} \subseteq P^{i+1}_{G,L}(S)$ in s tem smo dokazali tudi drugo inkluzijo.

Naj bo množica S optimalna L-električna dominantna množica. Ker je množica S tudi L-električna dominantna množica na grafu G natanko tedaj, ko je množica S tudi L'-električna dominantna množica na grafu G', dobimo $\gamma_{\mathrm{P},L}(G) = \gamma_{\mathrm{P},L'}(G')$. □

3.3 Implementacija

Pri implementaciji bomo potrebovali drevo, zato bomo najprej implementirali generator drevesa, nato bo pa sledil algoritem iz razdelka 3.2.

Majhnih dreves z nekaj vozlišči ni težko sprogramirati. Vozlišča in povezave lahko eksplicitno zakodiramo v kodo. Problem se pojavi, ko želimo zgraditi večje drevo z 100, 1000 ali več vozlišči. V tem primeru želimo imeti generator.

Algoritem, ki smo ga razvili za generiranje dreves, je naslednji:

```
ALGORITEM ZA GENERIRANJE DREVES
vhod: p - število vozlišč
izhod: G - drevo
begin
    dodaj v graf G začetno vozlišče v;
    while (število vozlišč ne doseže podanega števila vozlišč)
        dodaj v graf G novo vozlišče u;
        dodaj povezavo med u in poljubnim preostalim vozliščem v grafu G;
    end
end
```

Vhod v algoritem je parameter p, ki podaja red drevesa. V izhodu pa dobimo zgenerirano drevo G.

Pokažimo, da je rezultat algoritem res drevo. Ker je z vsako dodano povezavo na enem od krajišč dodano tudi novo vozlišče, ki nima drugih povezav, algoritem

nikoli ne sklene cikla. Ob ugotovitvi, da je graf tudi povezan, sledi, da je graf drevo. Še več! Graf je drevo na vsaki iteraciji zanke.

Zanima nas tudi, ali lahko z našim algoritmom za generiranje dreves zgeneriramo poljubno drevo. Odgovor je da!

Dokaz. Naj bo graf G poljubno drevo končne razsežnosti. Vsako drevo reda vsaj 2 ima vsaj en list, sicer graf vsebuje cikel in zato ni drevo. Če drevo torej ne bi imelo lista, bi bilo vsako vozlišče stopnje vsaj 2. Zato bi lahko na vsakem vozlišču pot nadaljevali. Ker je vozlišč končno mnogo, je nujno, da enkrat naletimo na vozlišče, ki je že na poti. To pomeni, da smo sklenili cikel.

Torej si na grafu G izberemo list x in pripadajočo povezavo e ter ju zbrišemo. Ker nismo sklenili nobenega cikla, je graf tudi sedaj drevo. Postopek rekurzivno ponavljamo, dokler nam ne ostane eno vozlišče.

Če postopek izvedemo v obratni smeri, na vsakem koraku dodamo list s povezavo. To pa je ravno naš zgoraj opisani algoritem za generiranje dreves. □

To pomeni, da lahko zgeneriramo poljubno drevo. Dodatno vprašanje pa je, ali imajo vsa drevesa reda n enako verjetnost generiranja. Iskanje odgovora prepuščamo bralcu, saj področje presega meje naše diplomske naloge.

Spodaj je prikazana implementacija generatorja drevesa v programskem jeziku Java.

```
1  public static Graph generateTree( int size ) {
2
3      Graph g = new SingleGraph("Graph");
4      Random rnd = new Random();
5      Node[] nodes = new Node[size];
6
7      Node root = g.addNode("0"); // dodajanje korena v seznam
8      nodes[0] = root; // dodajanje korena v seznam
9
10     // iterativno dodajanje vozlisc
11     for ( int i = 1; i < size; i++){
12
13       Node n = g.addNode(Integer.toString(i)); // novo vozlisce
14       nodes[i] = n; // dodajanje vozlisca v seznam
15       int rndNum = rnd.nextInt(i);  // nakljucno stevilo za dolocanje
     naklkjucnega soseda
16       Node parent = nodes[rndNum]; // dolocanje starsa
17
18       // dodajanje povezave med n (listom) in parent (starsem)
```

```
19        g.addEdge(  Integer.toString( i ) + "_" + Integer.toString( rndNum ) ,
      n, parent);
20      }
21      return g; // vrnemo generiran graf
22    }
```

Program 3.1: Funkcija generira drevo.

Funkcija *generateTree* v argumentu sprejme parameter *size*. Parameter pove funkciji, koliko vozlišč mora drevo vsebovati. Rezultat funkcije je objekt *Graph*, ki predstavlja drevo.

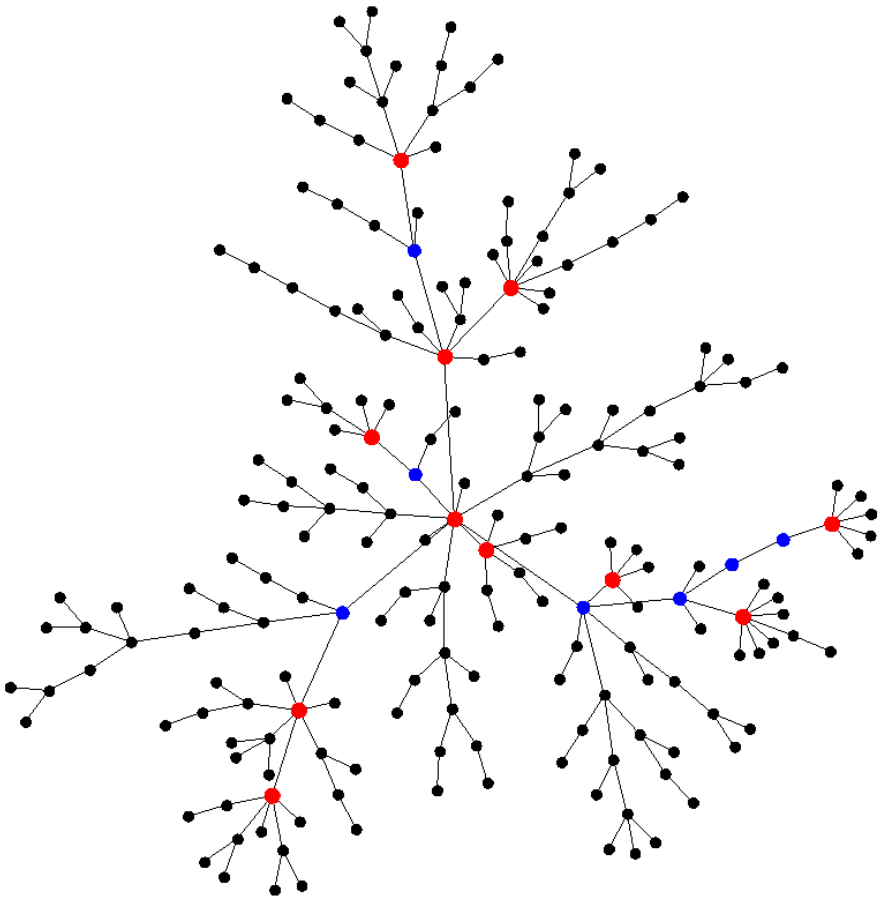
```
1  public static LinkedList<Node> treeAlgorithm( Graph g, int k_param ) {
2
3      k = k_param; // stopnja posplositve
4      LinkedList<Node> requiredList = new LinkedList<Node>(); // seznam
       vozlisc v stanju R
5      init(g); // zacetna inicializacija
6      ArrayList<Node> leaves = getLeaves(g); // pridobi seznam listov drevesa
7
8      while ( leaves.size() > 1 ) {
9        sleep(50);
10       Node n = leaves.remove(0); // odstrani prvi list v seznamu listov
11       System.out.println("Brisem list: "+n.getId());
12       Node parent = getParent(n); // pridobi starsa
13
14       String a_n = n.getAttribute(a); // pridobi oznako 'a_v' za list
15       int b_n = n.getAttribute(b) ; // pridobi oznako 'b_v' za list
16
17       String a_p = parent.getAttribute(a); // pridobi oznako 'a_v' za starsa
18       int b_p = (int)parent.getAttribute(b); // pridobi oznako 'b_v' za
       starsa
19
20       // dodatek za grafiko
21       if ( n.getAttribute("ui.class").equals("B0") ) n.setAttribute("ui.
       class", B);
22
23       // ALGORITEM
24       // (1)
25       if ( a_n.equals(R) ) {
26         if ( a_p.equals(B) ) {
27           parent.setAttribute(a, F);
28           parent.setAttribute("ui.class", F); // dodatek za grafiko
29         }
30       }
31       // (2)
32       else if ( a_p.equals(R) || ( a_n.equals(F) && b_n == 0)) {
```

```
33          // ni sprememb
34        }
35        // (3)
36        else if ( a_n.equals(B) && b_p > 0 ) {
37          parent.setAttribute("b", b_p - 1 );
38        }
39        // (4)
40        else if ( a_n.equals(B) && b_p == 0 ) {
41          parent.setAttribute(a, R);
42          parent.setAttribute("ui.class", R); // dodatek za grafiko
43          requiredList.add(parent); // vozlisce n je v stanju R, zato gre v
    requiredList
44        }
45        // (5)
46        else if ( a_n.equals(F) && b_n > 0 && !a_p.equals(R) ) {
47          parent.setAttribute(a, F);
48          parent.setAttribute("ui.class", F); // dodatek za grafiko
49        }
50        else
51          System.err.println("treeAlgorithm(): Napaka v algoritmu");
52
53        n.setAttribute(DELETED, true); // vozlisce n oznacimo za zbrisano;
    dodatek za grafiko
54
55        // iskanje stevila sosedov starsa potem, ko je list zbrisan
56        int neighbours = 0;
57        for ( Edge e : parent.getEachEdge() ) {
58          Node op = e.getOpposite(parent);
59          if ( (boolean)op.getAttribute(DELETED) == false ) {
60            neighbours++;
61          }
62        }
63
64        // preverjanje ali stars postane list v grafu G'
65        if ( neighbours == 1 ) {
66          leaves.add(parent); // dodaj vozlisce parent v seznam listov
67          System.out.println("Dodajam starsa med liste ...");
68          System.out.println(leaves);
69        }
70      }
71      return requiredList; // vrne seznam dominantnih vozlisc
72    }
```

Program 3.2: Funkcija z algoritmom za iskanje optimalne k-električne dominantne množice na drevesih.

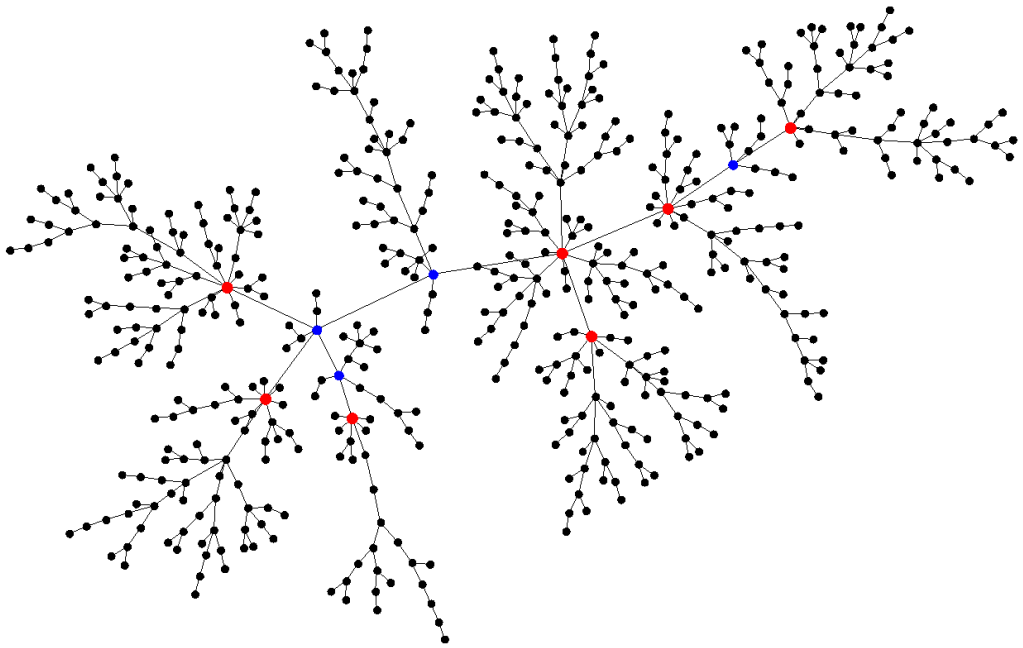
Funkcija *treeAlgorithm* v argumentih sprejme objekt *Graph*, ki mora biti

drevo, in parameter k_param, ki predstavlja stopnjo posplošenosti električne dominacije. Pred zanko izvedemo začetne nastavitve, medtem ko v zanki izvajamo algoritem 3.2. Pri implementaciji operacije izvajamo ves čas na istem grafu in spremembe zabeležimo v atributih knjižnice GraphStream. Operacija $G' = G - x$ iz algoritma je implementirana tako, da vozlišču x njegov atribut $DELETED$ nastavimo na *true* (vrstica 53). To se potem z ignoriranjem vozlišča upošteva pri njegovem sosedu oziroma staršu v vrstici 59.



Slika 3.1: Drevo reda 200. Z rdečo barvo so označena vozlišča v 3-električni dominantni množici. Z modro barvo so označena vozlišča v stanju F.

Na sliki 3.1 je prikazan rezultat funkcije *generateTree*, ki je zgradila drevo reda 200, in nato uporabe funkcije *treeAlgorithm*, ki je poiskala 3-električno dominantno množico. 3-električna dominantna množica je prikazana z rdečo barvo.



Slika 3.2: Drevo reda 500. Z rdečo barvo so označena vozlišča v 5-električni dominantni množici. Z modro barvo so označena vozlišča v stanju F.

Poglavje 4

Časovna zahtevnost

Problem iskanja električne dominantne množice je NP-poln problem. V tem poglavju bomo dokazali, da imamo NP-poln problem tudi, če se omejimo na dvodelne grafe.

Odločitveni problem P je NP-poln, kadar velja

(1) $P \in NP$,

(2) $P \in NP$-težek, kar pomeni, da lahko vsak problem iz prostora NP prevedemo nanj.

Definicija problema električne dominacije je naslednja.

Definicija 4.1. [5] *Naj bo podan graf G in pozitivno celo število $k > 1$. Ali ima graf G električno dominantno množico S, katere moč je $|S| \leq k$?*

V nadaljevanju bomo potrebovali še definicijo problema 3-SAT.

Definicija 4.2. [5] *Naj bo množica $U = \{u_1, u_2, \ldots, u_n\}$ množica spremenljivk $u_i \in \{true, false\}$. Naj bo stavek C_i množica treh literalov l_i, kjer je $l_i \in \{u_i, \overline{u_i}\}$. Naj bo stavek $C = \{C_1, C_2, \ldots, C_m\}$. Naj bo stavek C_i resničen takrat, ko je resničen vsaj en literal: $l_j = true$. Naj bo stavek C resničen, ko so resnični vsi stavki C_i. Ali obstaja prireditev vrednosti $true$ ali $false$ elementom u_i tako, da bo stavek C resničen?*

Velja naslednja trditev.

Izrek 4.3. [5] *Problem iskanja električne dominantne množice na dvodelnih grafih G je NP-poln problem.*

Dokaz. Označimo problem iskanja električne dominantne množice s Pe. Najprej bomo pokazali, da je $Pe \in NP$. To bo veljalo, ko bomo lahko rešitev problema Pe preverili v polinomskem času. Rešitev problem Pe je električna dominantna množica S. Veljavnost neenakosti $|S| \leq k$ lahko s sprehodom po elementih množice S preverimo v polinomskem času $O(|S|)$. V polinomskem času $O(V(G))$ pa lahko s sprehodom po vozliščih grafa G po pravilih P1 in P2 s podano električno dominantno množico preverimo tudi, ali množica S nadzoruje celoten graf. Torej smo dokazali $Pe \in NP$.

Za dokaz $Pe \in NP$-težek bomo uporabili poznan NP-poln problem: 3-SAT (iz definicije 4.2). Uporabili ga bomo za prevedbo na naš problem Pe. V dokazu bomo za stavke, literale in spremenljivke uporabili enake oznake kot v definiciji 4.1 in definiciji 4.2.

Zgradimo graf G iz spremenljivk u_i in stavkov C_j na naslednji način. Za vsako spremenljivko u_i zgradimo cikel R_i iz štirih vozlišč. Eno vozlišče dodelimo spremenljivki u_i, nasprotno vozlišče dodelimo komplementu spremenljivke u_i z oznako $\bar{u_i}$, preostali dve vozlišči pa označimo z A_i in B_i.

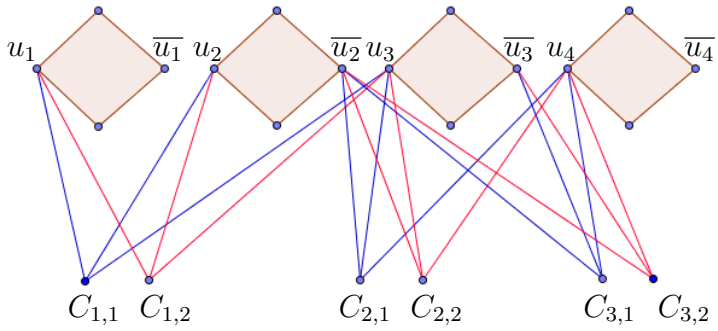
Za vsak stavek C_j dodamo na graf dve novi vozlišči in ju poimenujemo $C_{j,1}$ in $C_{j,2}$. Obe vozlišči povežemo s tistimi spremenljivkami iz množice U, ki so zastopane v literalih stavka C_j.

Primer grafa za $U = \{u_1, u_2, u_3, u_4\}$ in $C = \{\{u_1, u_2, u_3\}, \{\overline{u_2}, u_3, u_4\}, \{\overline{u_2}, \overline{u_3}, u_4\}\}$ prikazuje slika 4.1.

Pokažimo, da je stavek C resničen natanko tedaj, ko ima graf G električno dominantno množico moči največ $k = n$.

($\Rightarrow$) Naj bo stavek C resničen. Električno dominantno množico S na grafu G zgradimo po naslednjem pravilu. Če je spremenljivka $u_i = true$, potem jo vstavimo v množico S, sicer v množico S vstavimo $\overline{u_i}$. Pokažimo, da je množica S resnično električna dominantna množica.

V vsakem ciklu R_i je vsaj ena spremenljivka izmed u_i in $\overline{u_i}$ enaka $true$. Recimo, da je $u_i = true$, potem je spremenljivka u_i dodana v električno dominantno množico S. Po pravilu P1 sta v nadzorovano množico dodani tudi vozlišči A_i ter B_i. Obema vozliščema ostane eno nenadzorovano vozlišče ($\overline{u_i}$), zato izpolnjujeta



Slika 4.1: Graf, ki je generiran iz 3-SAT problema.

pogoj širjenja nadzora. Torej je nadzorovano tudi vozlišče $\overline{u_i}$ in s tem je celoten cikel R_i nadzorovan. Enako dobimo, če nastavimo $\overline{u_i} = true$.

Po predpostavki je vsak stavek C_j resničen, zato obstaja tudi literal l stavka C_j, ki je resničen. Ker je l resničen, je dodan v dominantno množico S. Zato sta po pravilu P1 tudi vozlišči $C_{j,1}$ in $C_{j,2}$ nadzorovani.

Torej električna dominantna množica S nadzoruje celoten graf.

($\Leftarrow$) Sedaj pa vzemimo električno dominantno množico moči največ $k = n$. Dokažimo, da mora potem biti stavek C resničen.

V dominantni množici S mora biti vsaj eno vozlišče iz vsakega cikla R_i. Sicer vozlišči A_i in B_i ne bi bili nadzorovani, saj bi vozlišči u_i in $\overline{u_i}$ imeli vsaj dve sosednji vozlišči nenadzorovani (A_i in B_i) in ne bi izpolnjevali širitvenega pogoja P2.

Število ciklov R_i je enako n, zato je $|S| \geq n$. Ker smo predpostavili, da je $|S|$ največ $k = n$, sledi $|S| = n$.

Električna dominantna množica S ima vse svoje elemente iz ciklov R_i. Še več! Iz vsakega cikla ima natanko enega. Po predpostavki električna dominantna množica S nadzoruje celoten graf. Zato so nadzorovani tudi vsi stavki $C_{j,*}$ za vsak j in $*$, kjer je $* \in \{1, 2\}$. Vsak stavek je nadzorovan s strani kakšnega literala iz množice S. Če to ne bi veljalo za kakšen stavek $C_{x,*}$ (opomnimo, da je $C_{x,1}$ nadzorovan natanko tedaj, ko je nadzorovan $C_{x,2}$, saj sta povezana z istimi literali), bi bil ta stavek nadzorovan iz nekega dominantnega vozlišča A_i (enako je za B_i). Ker pa ima spremenljivka u_i (oziroma $\overline{u_i}$, če je stavek povezan s to spremenljivko)

dva nenadzorovana soseda $C_{x,1}$ in $C_{x,2}$, ne more širiti nadzora. Tako pridemo v protislovje.

Torej so resnični vsi stavki. S tem zaključimo dokaz.

□

Dokazali smo, da je iskanje električne dominantne množice NP-poln problem na dvodelnih grafih. Ker so dvodelni grafi podmnožica splošnih grafov, enako velja za splošne grafe.

Ker je električna dominacija poseben primer k-električne dominacije, je NP-poln problem tudi k-električna dominacija.

Za konec si oglejmo definicijo grafov Sierpińskega. Grafi Sierpińskega so poleg dreves do zdaj edini znani netrivialni razred grafov, za katere je problem k-električne dominantne množice polinomske časovne zahtevnosti.

Za $n \in \mathbb{N}$ definirajmo množici $[n]_0 = \{0, \ldots, n-1\}$ in $[n] = \{1, \ldots, n\}$.

Definicija 4.4. [3] *Naj bosta $n, p \in \mathbb{N}$. Množica vozlišč grafa Sierpińskega S_p^n je množica $[p]_0^n$, pri kateri vozlišče označimo s $s_{n-1} \ldots s_0$, $s_i \in [p]_0$. Vozlišči $s = s_{n-1} \ldots s_0$ in $t = t_{n-1} \ldots t_0$ sta sosednji natanko tedaj, ko obstaja $\delta \in [n]$ tako, da velja:*

(1) $s_d = t_d$ za $d \in [n] \backslash [\delta]$;

(2) $s_\delta \neq t_\delta$;

(3) $s_d = t_\delta$ in $t_d = s_\delta$ za $d \in [\delta - 1]$.

Tabela 4.1 primerja časovno kompleksnost iskanja optimalne dominantne množice in optimalne električne dominantne množice na različnih grafih. Pri dvodelnih, tetivnih, cikličnih, ravninskih in razcepljenih grafih je problem v obeh primerih NP-poln. Na drevesih in grafih Sierpińskega pa lahko optimalno dominantno množico in tudi električno dominantno množico poiščemo v polinomskem času.

Razredi grafov	Dominantna množica	Električna dominantna množica
Dvodelni	NP-poln	NP-poln
Tetivni	NP-poln	NP-poln
Ciklični	NP-poln	NP-poln
Ravninski	NP-poln	NP-poln
Razcepljeni	NP-poln	NP-poln
Drevesa	P	P
Grafi Siérpinskega	P	P

Tabela 4.1: Časovna zahtevnost posameznih grafov pri iskanju optimalne dominantne in optimalne električne dominantne množice. Podatki za grafe Sierpińskega so pridobljeni iz [3], podatki preostalih grafov pa iz [4].

Poglavje 5

Zaključek

V diplomski nalogi smo predstavili problem osnovne in posplošene električne dominacije ter pokazali, da sta problema na drevesih rešljiva v polinomskem času. S pomočjo knjižnice GraphStream smo postopek iskanja k-električne dominantne množice na drevesih tudi vizualizirali. Dokazali smo, da je poljubna k-električna dominantna množica na grafu G tudi k'-električna dominantna množica, kjer je $k' > k$. Nato smo pokazali, da je $\frac{n}{k+2}$ zgornja meja za k-električno dominantno število in v poglavju časovne zahtevnosti na dvodelnih grafih dokazali še NP-polnost problema osnovne električne dominacije.

Svet je danes poln tehnoloških inovacij, vendar to ne ustavi nadaljnega razvoja in novih idej. Številna brezžična omrežja, ki omogočajo brezžičen prenos slike, zvoka in ostalih podatkov, so v polnem razmahu. V nekaterih električnih zobnih ščetkah pa se uporablja celo brezžični prenos električne energije.

Mogoče se lahko z razvojem brezžičnih električnih omrežij izognemo problemu osnovne in posplošene električne dominacije, verjetno pa bo na to potrebno počakati še kar nekaj časa. Problem osnovne in posplošene električne dominacije vsaj za-enkrat ostaja aktualen.

Literatura

[1] A. Aazami, Hardness results and approximation algorithms for some problems on graphs, 2008, Doktorska disertacija, University of Waterloo, 2008.

[2] G.J. Chang, P. Dorbec, M. Montassier, A. Raspaud, Generalized Power Domination of Graphs, Discrete Appl. Math. 160 (2012) 1691–1698.

[3] P. Dorbec, S. Klavžar, Generalized Power Domination: Propagation Radius and Sierpiński Graphs, Acta Appl. Math., v tisku, DOI: 10.1007/s10440-014-9870-7.

[4] J. Guo, R. Niedermeier, D. Raible, Improved Algorithms and Complexity Results for Power Domination in Graphs, Algorithmica 52 (2008) 177–202.

[5] T.W. Haynes, S.M. Hedetniemi, S.T. Hedetniemi, M.A. Henning, Domination in Graphs Applied to Electric Power Networks, SIAM J. Discrete Math. 15 (2002) 519–529.

[6] http://en.wikipedia.org/wiki/Alessandro_Volta, dostop 15-8-2014.

[7] http://en.wikipedia.org/wiki/Electric_power_industry, dostop 15-8-2014.

[8] http://sl.wikipedia.org/wiki/Elektrika, dostop 15-8-2014.

Dodatek A

V dodatku so priloženi celotni Java razredi za programe v poglavju 2.4 in poglavju 3.3.

```
1  import org.graphstream.graph.Graph;
2  import org.graphstream.graph.Node;
3  import org.graphstream.graph.implementations.SingleGraph;
4
5  public class ConnectedStars {
6
7    public static int id = 0;
8
9    public static void main(String[] args) {
10
11     // nenarascujoce zaporedje pozitivnih celih stevil
12     int[] seq = {7, 5, 5, 3, 2, 0};
13     int k = 3;
14
15     // ustvari graf in ga inicializira
16     Graph g = new SingleGraph("Graph");
17     g.addAttribute("ui.stylesheet", styleSheet);
18     g.display();
19
20     Node[] nodes = createClique(g, seq[0]); // funkcija naredi kliko in vrne
        seznam vozlisc
21     colorizeClique (nodes); // funkcija pobarva vozlisca v kliki modro
22     compute(g, seq, nodes, k); // funkcija kreira zvezde, kjer so centri
       zvezd vozlisca iz klike
23   }
24
25
26   public static void createStar ( Graph g, Node root, int length, int
       density ) {
27     for ( int i = 0; i < density; i++ ) {
28       Node parent = root;  // parent je vozlisce, ki bo dobilo novega soseda
29       for ( int j = 0; j < length; j++ ) {
30         Node child = g.addNode( Integer.toString( id += 1 ) ); // nov sosed
```

```
31          g.addEdge( Integer.toString( id += 1 ), parent, child ); // nova
    povezava
32          parent = child; // nastavi novo vozlisce
33        }
34      }
35    }
36
37    public static Node[] createClique ( Graph g, int size ) {
38
39      Node[] nodes = new Node[size];
40      for ( int i = 0; i < size; i++ ) {
41        Node n = g.addNode( Integer.toString( id += 1 ) ); // ustvari vozlisce
42        nodes[i] = n; // dodaj v seznam
43        for ( int j = 0; j < i; j++ ) { // dodaj povezave s preostalimi
    vozlisci
44          g.addEdge( Integer.toString( id += 1 ), nodes[j], n ); // dodaj
    povezovo
45        }
46      }
47      return nodes;
48    }
49
50    public static void compute ( Graph g, int[] seq, Node[] nodes, int domNum
    ) {
51      int counter = 0;
52      // zanka po razlicnih vrstah zvezd
53      for ( int k = 0; k < seq.length - 1; k++ ) {
54        int dif = seq[k] - seq [k+1];
55        // kopije zvezd z enakim stevilom listov
56        for ( int j = 0; j < dif; j++ ) {
57          createStar( g, nodes[counter+j], 1, k+1 ); // funkcija, ki ustvari
    zvezdo
58          // pobarvaj center zvezde, ce le to ustreza pogoju
59          if ( k>= domNum ) {
60            nodes[counter+j].setAttribute( "ui.class", "required" );
61          }
62        }
63        // prestavi counter na zacetek naslednjega sklopa centrov
64        counter += dif;
65      }
66    }
67
68    public static void colorizeClique ( Node[] nodes ) {
69
70      for ( Node n : nodes ) {
71        n.setAttribute("ui.class", "marked");
72      }
```

```
73   }
74
75   protected static String styleSheet =
76       "node {" +
77         "    fill-color: black;" +
78         "}" +
79         "node.marked {" +
80         "    fill-color: blue;" +
81         "size:15px;" +
82         "}" +
83         "node.required {" +
84         "    fill-color: red;"
85         + "size:15px;" +
86         "}";
87 }
```

Program 5.1: ConnectedStars.java.

```
1  import org.graphstream.graph.Graph;
2  import org.graphstream.graph.implementations.SingleGraph;
3
4  public class RunTree {
5
6    public static int k = 3;     // elektricno dominantno stevilo
7    public static int size = 200; // stevilo vozlisc
8
9    public static void main(String[] args) {
10
11     Graph g = new SingleGraph("Graph");
12     g.display();
13
14     RandomTreeGenerator.generateTree(g, size);
15     g.addAttribute("ui.stylesheet", TreeAlgorithm.styleSheet);
16     TreeAlgorithm.treeAlgorithm(g, k);
17   }
18 }
```

Program 5.2: RunTree.java.

```
1  import java.util.Random;
2  import org.graphstream.graph.Graph;
3  import org.graphstream.graph.Node;
4
5  public class RandomTreeGenerator {
6
7  public static Graph generateTree( int size ) {
8
```

```
9       Graph g = new SingleGraph("Graph");
10      Random rnd = new Random();
11      Node[] nodes = new Node[size];
12
13      Node root = g.addNode("0"); // dodajanje korena v seznam
14      nodes[0] = root; // dodajanje korena v seznam
15
16      // iterativno dodajanje vozlisc
17      for ( int i = 1; i < size; i++){
18
19        Node n = g.addNode(Integer.toString(i)); // novo vozlisce
20        nodes[i] = n; // dodajanje vozlisca v seznam
21        int rndNum = rnd.nextInt(i);  // nakljucno stevilo za dolocanje
      naklkjucnega soseda
22        Node parent = nodes[rndNum]; // dolocanje starsa
23
24        // dodajanje povezave med n (listom) in parent (starsem)
25        g.addEdge(  Integer.toString( i ) + "_" + Integer.toString( rndNum ) ,
       n, parent);
26      }
27      return g; // vrnemo generiran graf
28    }
29  }
```

Program 5.3: RandomTreeGenerator.java.

```
1   import java.util.ArrayList;
2   import java.util.LinkedList;
3   import org.graphstream.graph.Edge;
4   import org.graphstream.graph.Graph;
5   import org.graphstream.graph.Node;
6
7   public class TreeAlgorithm {
8
9     public static final String R = "R";
10    public static final String B = "B";
11    public static final String F = "F";
12    public static final String a =  "a";
13    public static final String b =  "b";
14    public static final String DELETED = "DELETED";
15    public static int k = 3;
16
17    public static LinkedList<Node> treeAlgorithm( Graph g, int k_param ) {
18
19      k = k_param; // stopnja posplositve
20      LinkedList<Node> requiredList = new LinkedList<Node>(); // seznam
      vozlisc v stanju R
21      init(g); // zacetna inicializacija
```

```
22      ArrayList<Node> leaves = getLeaves(g); // pridobi seznam listov drevesa
23
24      while ( leaves.size() > 1 ) {
25        sleep(50);
26        Node n = leaves.remove(0); // odstrani prvi list v seznamu listov
27        System.out.println("Brisem list: "+n.getId());
28        Node parent = getParent(n); // pridobi starsa
29
30        String a_n = n.getAttribute(a); // pridobi oznako 'a_v' za list
31        int b_n = n.getAttribute(b) ; // pridobi oznako 'b_v' za list
32
33        String a_p = parent.getAttribute(a); // pridobi oznako 'a_v' za starsa
34        int b_p = (int)parent.getAttribute(b); // pridobi oznako 'b_v' za
      starsa
35
36        // dodatek za grafiko
37        if ( n.getAttribute("ui.class").equals("B0") ) n.setAttribute("ui.
      class", B);
38
39        // ALGORITEM
40        // (1)
41        if ( a_n.equals(R) ) {
42          if ( a_p.equals(B) ) {
43            parent.setAttribute(a, F);
44            parent.setAttribute("ui.class", F); // dodatek za grafiko
45          }
46        }
47        // (2)
48        else if ( a_p.equals(R) || ( a_n.equals(F) && b_n == 0)) {
49          // ni sprememb
50        }
51        // (3)
52        else if ( a_n.equals(B) && b_p > 0 ) {
53          parent.setAttribute("b", b_p - 1 );
54        }
55        // (4)
56        else if ( a_n.equals(B) && b_p == 0 ) {
57          parent.setAttribute(a, R);
58          parent.setAttribute("ui.class", R); // dodatek za grafiko
59          requiredList.add(parent); // vozlisce n je v stanju R, zato gre v
      requiredList
60        }
61        // (5)
62        else if ( a_n.equals(F) && b_n > 0 && !a_p.equals(R) ) {
63          parent.setAttribute(a, F);
64          parent.setAttribute("ui.class", F); // dodatek za grafiko
65        }
```

```
66        else
67          System.err.println("treeAlgorithm(): Napaka v algoritmu");
68
69        n.setAttribute(DELETED, true); // vozlisce n oznacimo za zbrisano;
      dodatek za grafiko
70
71        // iskanje stevila sosedov starsa potem, ko je list zbrisan
72        int neighbours = 0;
73        for ( Edge e : parent.getEachEdge() ) {
74          Node op = e.getOpposite(parent);
75          if ( (boolean)op.getAttribute(DELETED) == false ) {
76            neighbours++;
77          }
78        }
79
80        // preverjanje ali stars postane list v grafu G'
81        if ( neighbours == 1 ) {
82          leaves.add(parent); // dodaj vozlisce parent v seznam listov
83          System.out.println("Dodajam starsa med liste ...");
84          System.out.println(leaves);
85        }
86      }
87      return requiredList; // vrne seznam dominantnih vozlisc
88    }
89
90    // inicializacija
91    private static void init (Graph g) {
92
93      for( Node n : g.getEachNode() ) {
94        n.setAttribute(a, B); // nastavi labelo a
95        n.setAttribute("ui.class", "B0"); // dodatek za grafiko
96        n.setAttribute("b", n.getDegree() < k ? n.getDegree() : k); // nastavi
       labelo b
97        n.setAttribute(DELETED, false); // vozlisce ni zbrisano; dodatek za
      grafiko
98      }
99      System.out.println("Inicializacija koncana ...");
100   }
101
102   // pridobi seznam listov grafa g
103   private static ArrayList<Node> getLeaves(Graph g) {
104
105     ArrayList<Node> a = new ArrayList<Node>();
106     for ( Node n : g.getEachNode() ) {
107       if ( n.getDegree() == 1 )
108         a.add(n);
109     }
```

```
110     System.out.println("Listi pridobljeni ...");
111     return a;
112   }
113
114   public static void sleep(int n) {
115     try {
116       Thread.sleep(n);
117     } catch (InterruptedException e) {
118       e.printStackTrace();
119     }
120   }
121
122   // pridobi prvega nezbrisanega starsa vozlisca n
123   public static Node getParent (Node n) {
124     for ( Edge e : n.getEachEdge() ) {
125       Node parent = e.getOpposite(n);
126       if ( (boolean)parent.getAttribute(DELETED) == false )
127         return parent;
128     }
129     System.out.println("Vozlisce nima starsev.");
130     return null;
131   }
132
133   protected static String styleSheet =
134       "node {" +
135         "    fill-color: #00CC00;" +
136         "size:13px;" +
137         "}" +
138         "node.R {" +
139         "    fill-color: red;" +
140         "size:17px;" +
141         "}" +
142         "node.B0 {" +
143         "    fill-color: #00CC00;" +
144         "}" +
145         "node.F {" +
146         "    fill-color: blue;" +
147         "size:11px;" +
148         "}"+
149         "node.B {" +
150         "    fill-color: #EB99FF;" +
151         "size:7px;" +
152         "}";
153 }
```

Program 5.4: TreeAlgorithm.java.